

VHDL, Verilog, and the Altera environment Tutorial

Table of Contents

1. Create a new Project
 2. Example Project 1: Full Adder in VHDL
 3. Code Compilation
 4. Pin Assignment
 5. Simulating the Designed Circuit
 6. Programming and Configuring the FPGA Device
 7. Example Project 2: Full Adder in Verilog
 8. Lab 1 Assignment
 9. Lab Report Guidelines
- Appendix A: VHDL and Verilog Standard Formats

This tutorial is intended to familiarize you with the Altera environment and introduce the hardware description languages VHDL and Verilog. The tutorial will step you through the implementation and simulations of a full-adder in both languages. Using this background you will implement a four-bit adder in both VHDL and Verilog. In the future, HDL labs can be done in either language.

You may want to refer to Appendix A to review the standard structures of VHDL and Verilog modules.

1. Create a new Project

On starting Altera Quartus II, you should be faced with a screen like this:

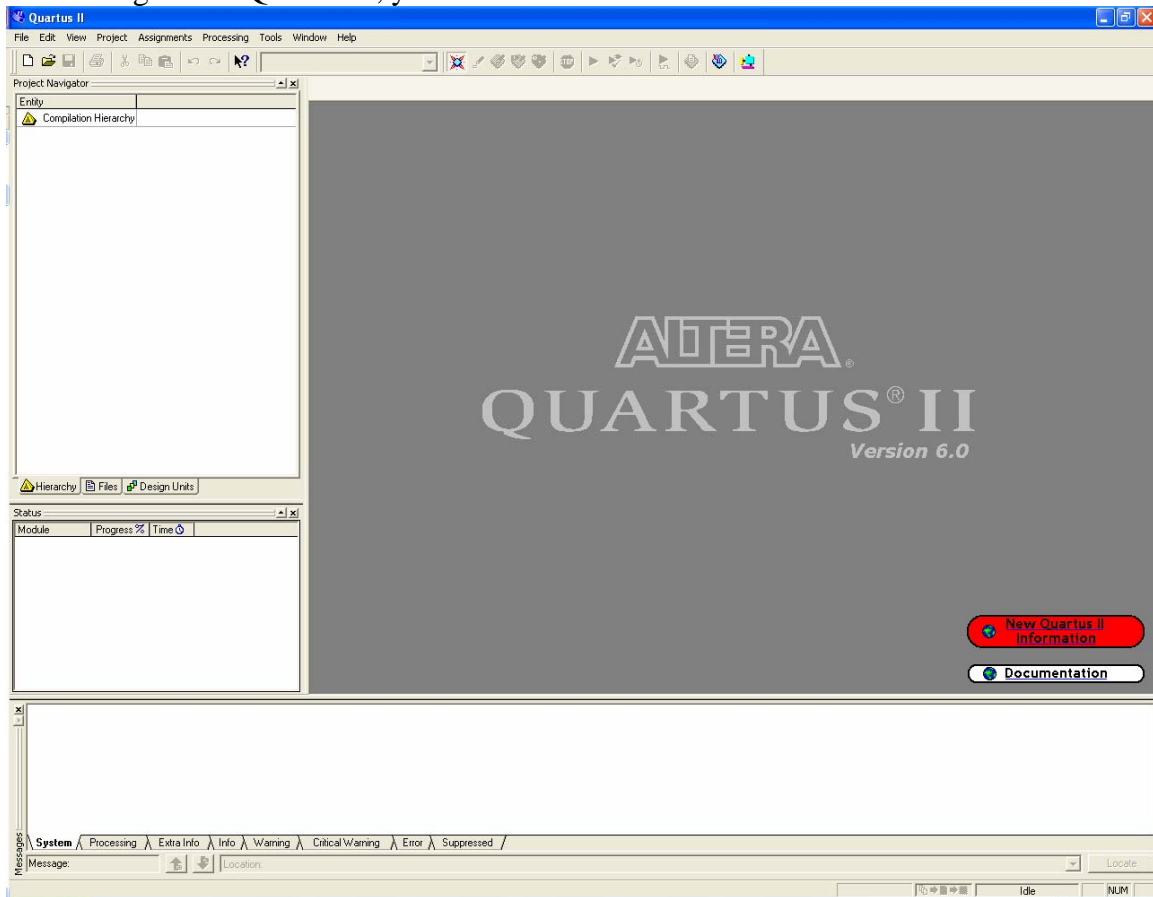


Figure 1. The main Quartus II display.

Go to "**File -> New Project Wizard**". A introduction Dialog will appear (Fig 2), It indicates the capability of this wizard. You can skip this window in subsequent projects by checking the box **Don't show me this introduction again**.

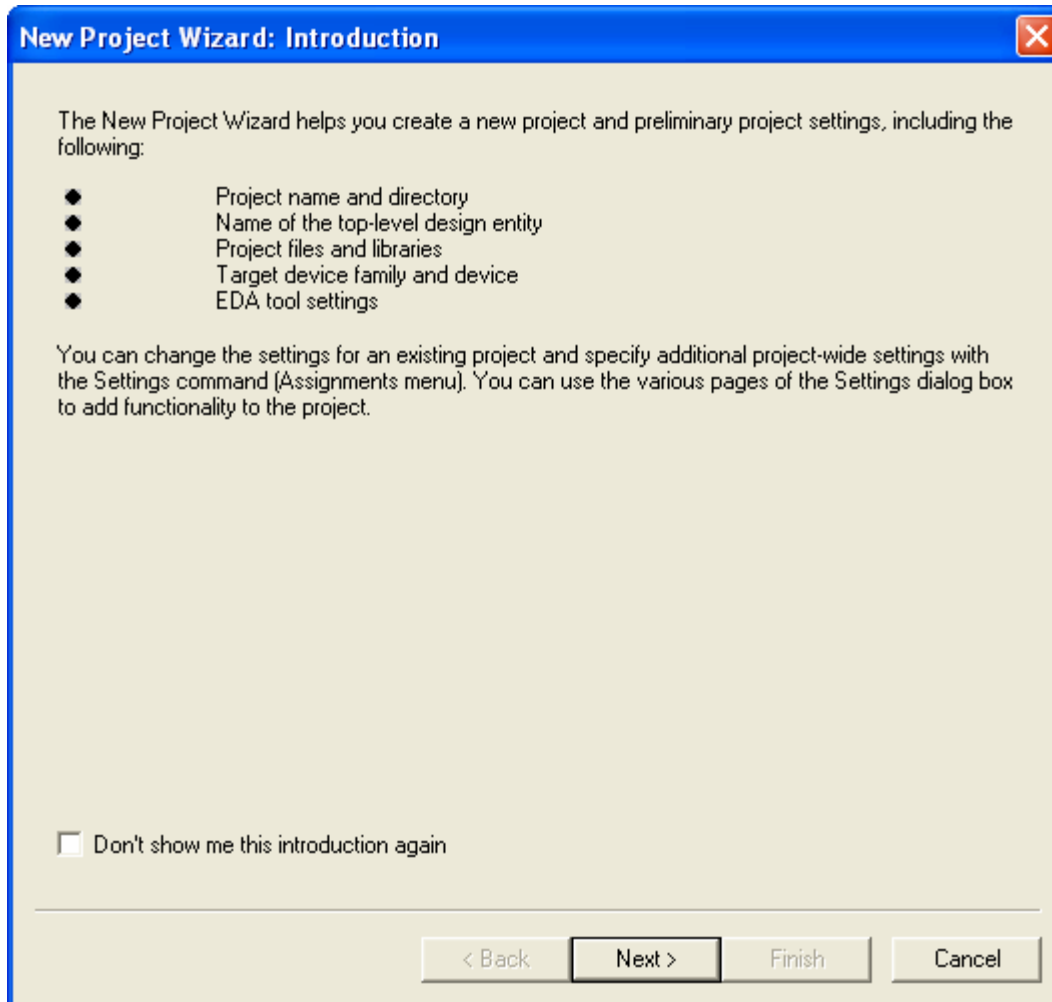


Figure. 2 Tasks performed by the wizard.

Press **Next** to get the window shown in Figure 3. Choose the location of your working directory and type in the name of your project (let's use *fulladder*) as shown in Fig. 3.

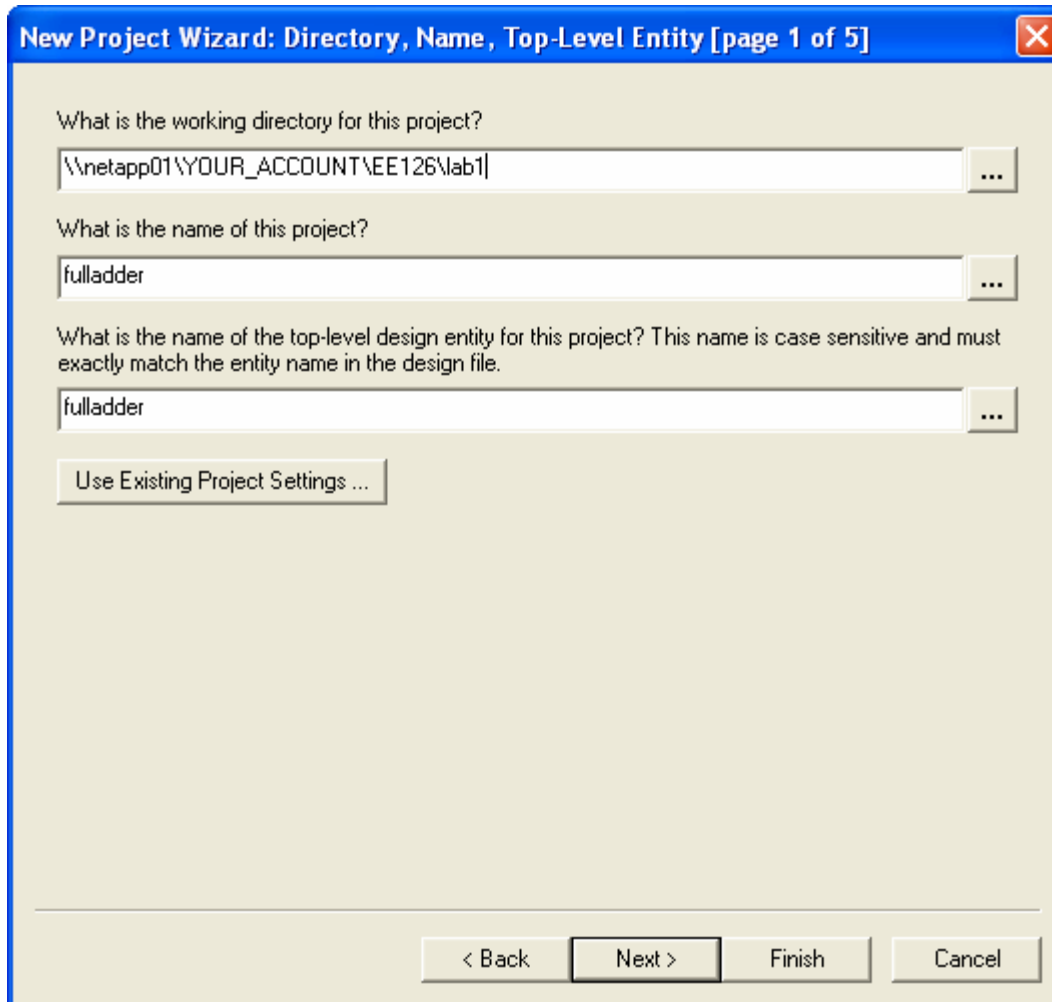


Figure. 3 Creation of a new project.

Press **Next**. Since we have not yet created the directory *lab1*, Quartus II software displays the pop-up box in Figure 4 asking if it should create the desired directory.

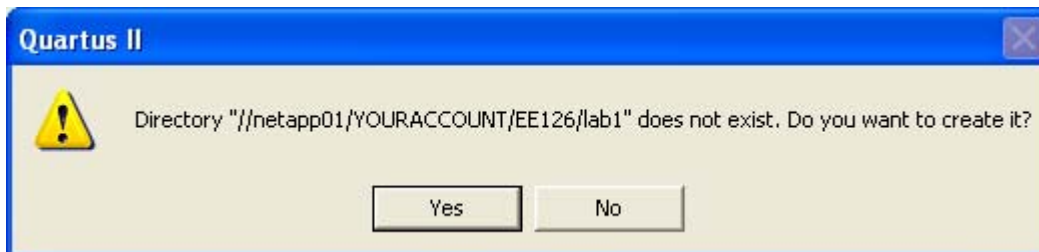


Figure. 4 Quartus II software can create a new directory for the project.

Click **Yes**, which leads to the windows in Figure 5.

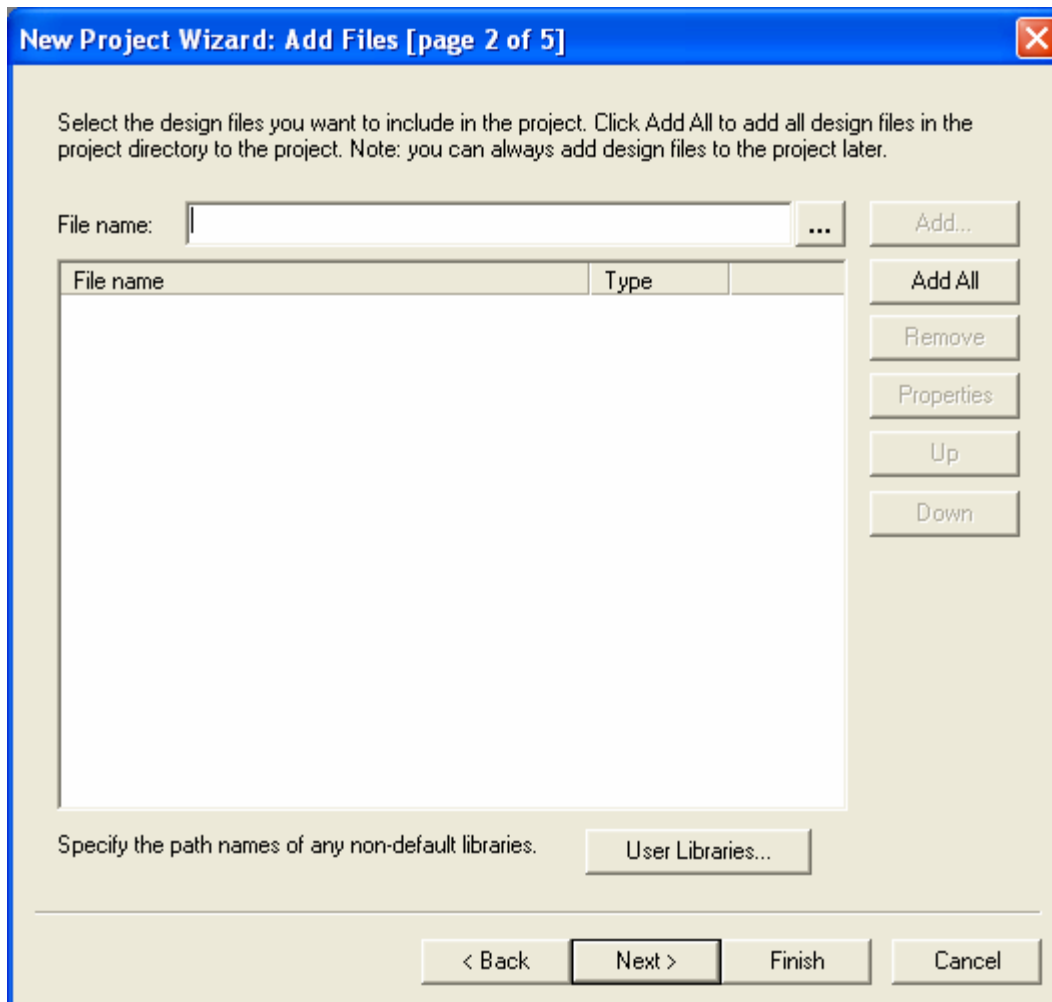


Figure 5. The wizard can include user-specified design files.

The wizard makes it easy to specify which existing files (if any) should be included in the project. Assuming that we do not have any existing files, click **Next**, which leads to the window in Figure 6

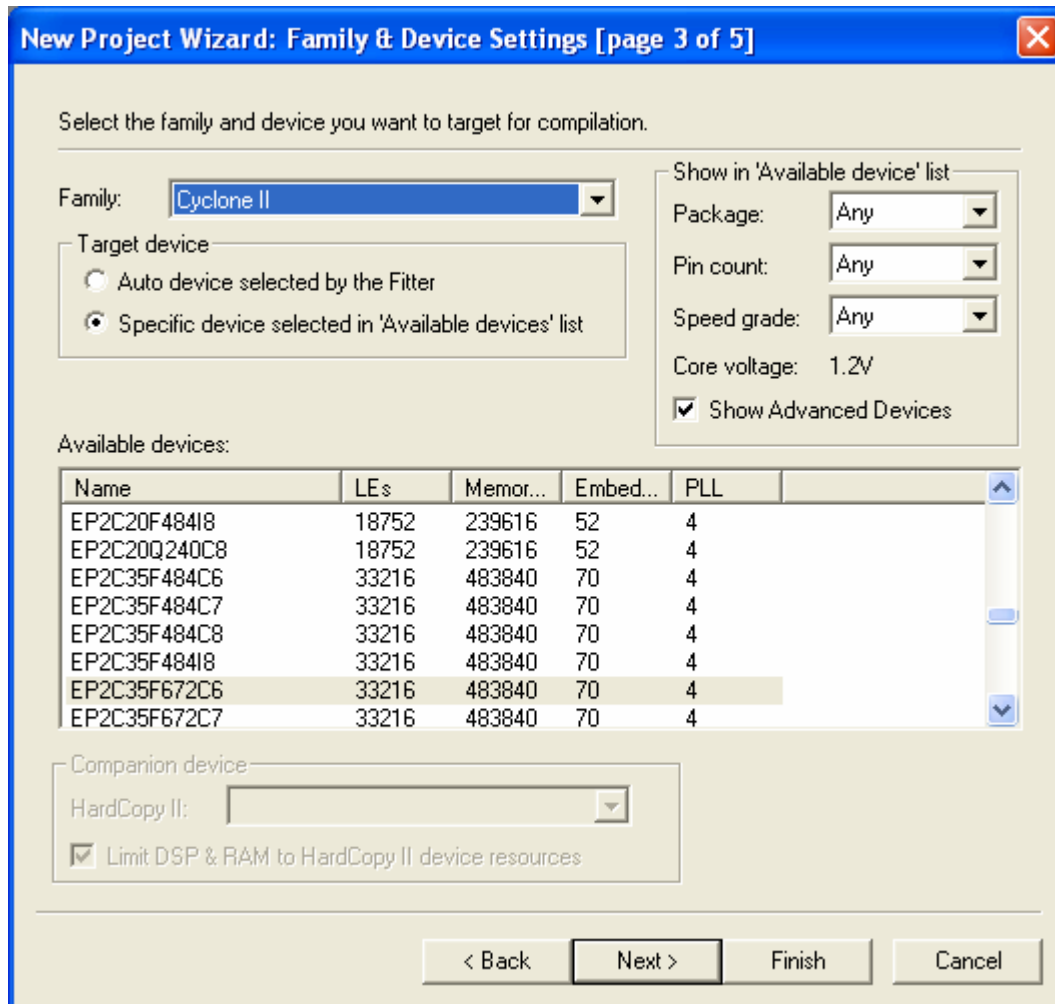


Figure 6. Choose the device family and a specific device.

We have to specify the type of device in which the designed circuit will be implemented. **Choose Cyclone™ II** as the target device family. We can let Quartus II software select a specific device in the family, or we can choose the device explicitly. We will take the latter approach. From the list of available devices, choose the device called **EP2C35F672C6** which is the FPGA used on Altera's DE2 board. Press **Next**, which opens the window in Figure 7.

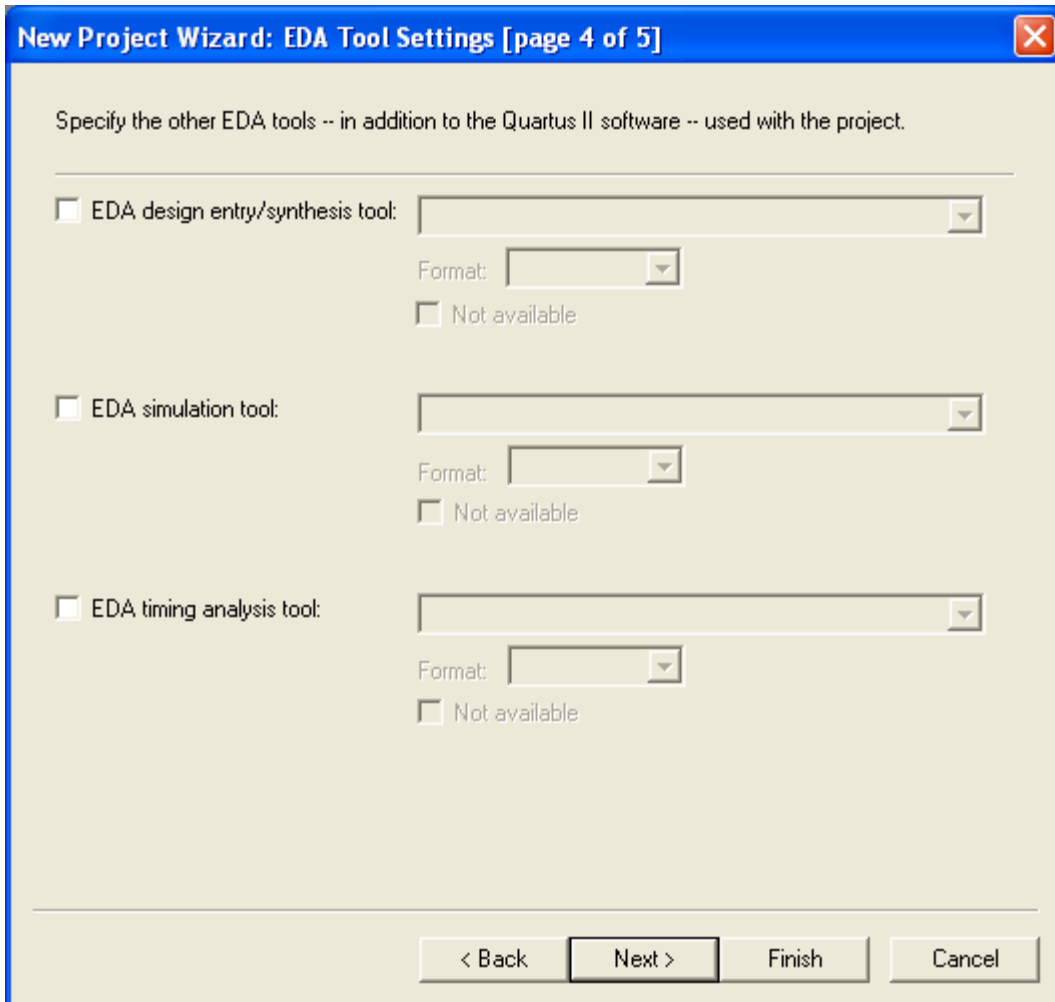


Figure 7. Other EDA tools can be specified.

The user can specify any third-party tools that should be used. A commonly used term for CAD software for electronic circuits is *EDA tools*, where the acronym stands for Electronic Design Automation. This term is used in Quartus II messages that refer to third-party tools, which are the tools developed and marketed by companies other than Altera. Since we will rely solely on Quartus II tools, we will not choose any other tools. Press **Next**.

A summary of the chosen settings appears in the screen shown in Figure 8. Press **Finish**, which returns to the main Quartus II window, but with *lab1_YOURNAME* specified as the new project, in the display title bar.

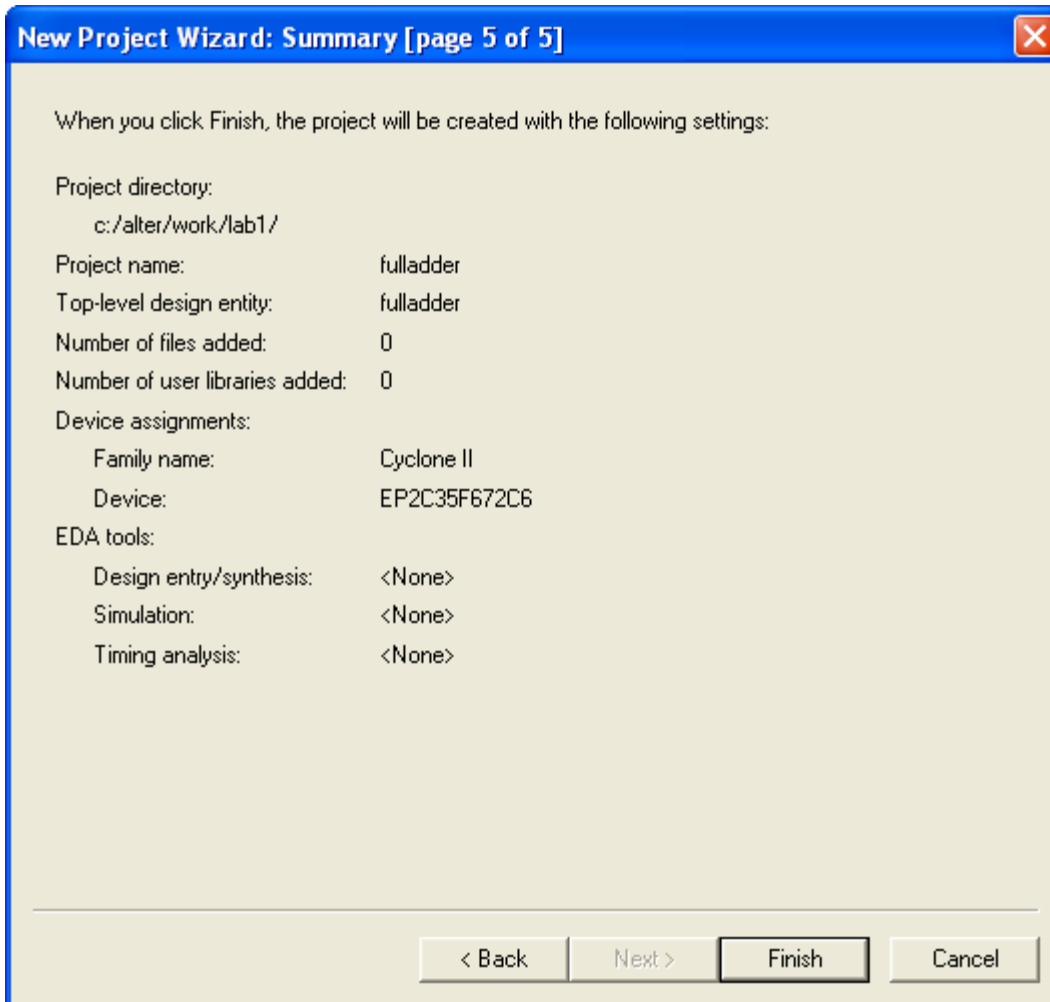


Figure 8. Summary of the project settings.

2. Example Project 1: Full Adder in VHDL

Select “**File > New**” to get the window in Figure 9, choose **VHDL File**, and click **OK**. This opens the Text Editor window. The first step is to specify a name for the file that will be created. Select **File > Save As** to open the pop-up box depicted in Figure 10. In the box labeled **Save as type** choose **VHDL File**. In the box labeled **File name** enter *fulladder*. Put a checkmark in the box **Add file to current project**. Click **Save**, which puts the file into the directory lab1. Maximize the Text Editor window and enter the VHDL code as shown in Figure 11. Save the file by typing **File > Save**, or by typing the shortcut **Ctrl-s**.

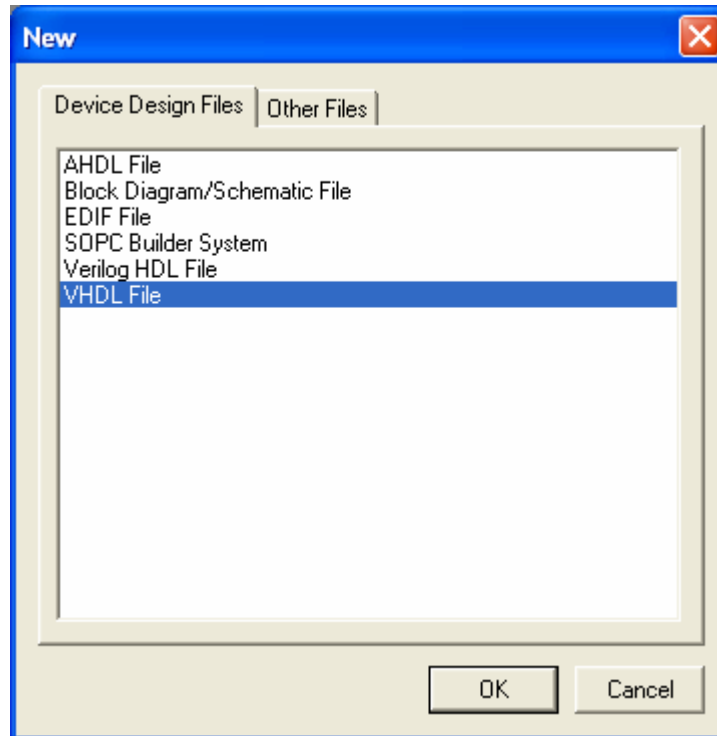


Figure 9. Choose to prepare a VHDL file.

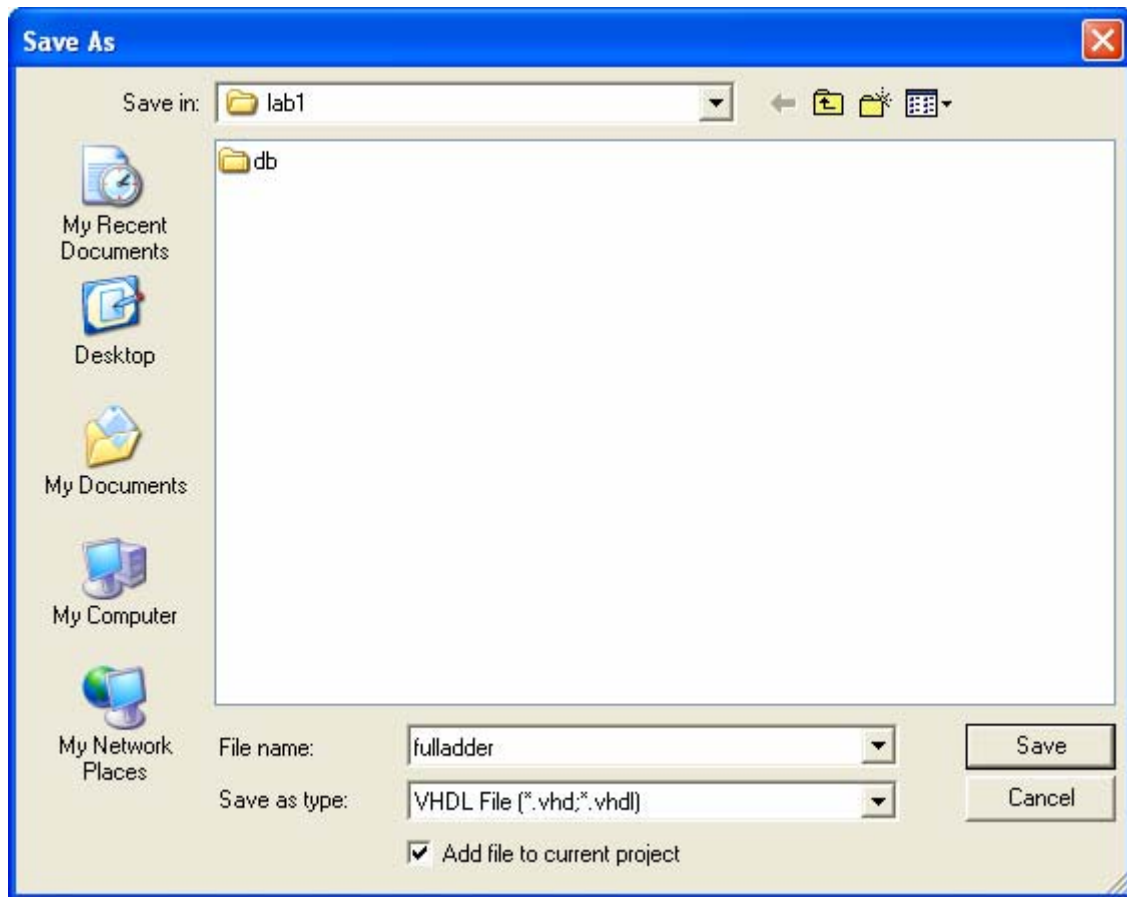


Figure 10. Name the file

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity fulladder is
  port ( a:  in std_logic;
         b:  in std_logic;
         cin: in std_logic;
         sum: out std_logic;
         cout: out std_logic);
end fulladder;

architecture Behavioral of fulladder is
  signal s1,s2,s3: std_ulogic;
  constant gate_delay: Time :=100 ps;
begin
  s1<=(a xor b) after gate_delay;
  s2<=(cin and s1) after gate_delay;
  s3<=(a and b) after gate_delay;
  sum<=(s1 xor cin) after gate_delay;
  cout<=(s2 or s3) after gate_delay;
end Behavioral;
```



Figure 11. fulladder VHDL code.

NOTE:

- **Constant can be used to declare a constant of a particular type. In this case, Time.**
- **The functional relation between the input and output signals is described by the architecture body.**
- **Only one architecture body should be bound to an entity, although many architecture bodies can be defined.**

The syntax of VHDL code is sometimes difficult for a designer to remember. To help with this issue, the Text Editor provides a collection of *VHDL templates*. The templates provide examples of various types of VHDL statements, such as an *ENTITY* declaration, a *CASE* statement, and assignment statements. It is worthwhile to browse through the templates by selecting **Edit > Insert Template > VHDL** to become familiar with this resource.

3. Code Compilation

The code in the file *fulladder* is processed by several **Quartus II** tools that analyze the code, synthesize the circuit, and generate an implementation of it for the target chip. These tools are controlled by the application program called the **Compiler**. Run the **Compiler** by selecting **Processing > Start Compilation**, or by clicking on the toolbar icon  that looks like a purple triangle. As the compilation moves through various stages, its progress is reported in a window on the left side of the Quartus II display. Successful (or unsuccessful) compilation is indicated in a pop-up box. Acknowledge it by clicking **OK**, which leads to the Quartus II display in Figure 12. In the message window, at the bottom of the figure, various messages are displayed. In case of errors, there will be appropriate messages given. When the compilation is finished, a compilation report is produced. A window showing this report is opened automatically, as seen in Figure 12. The window can be resized, maximized, or closed in the normal way, and it can be opened at any time either by selecting **Processing > Compilation Report** or by clicking on the icon .

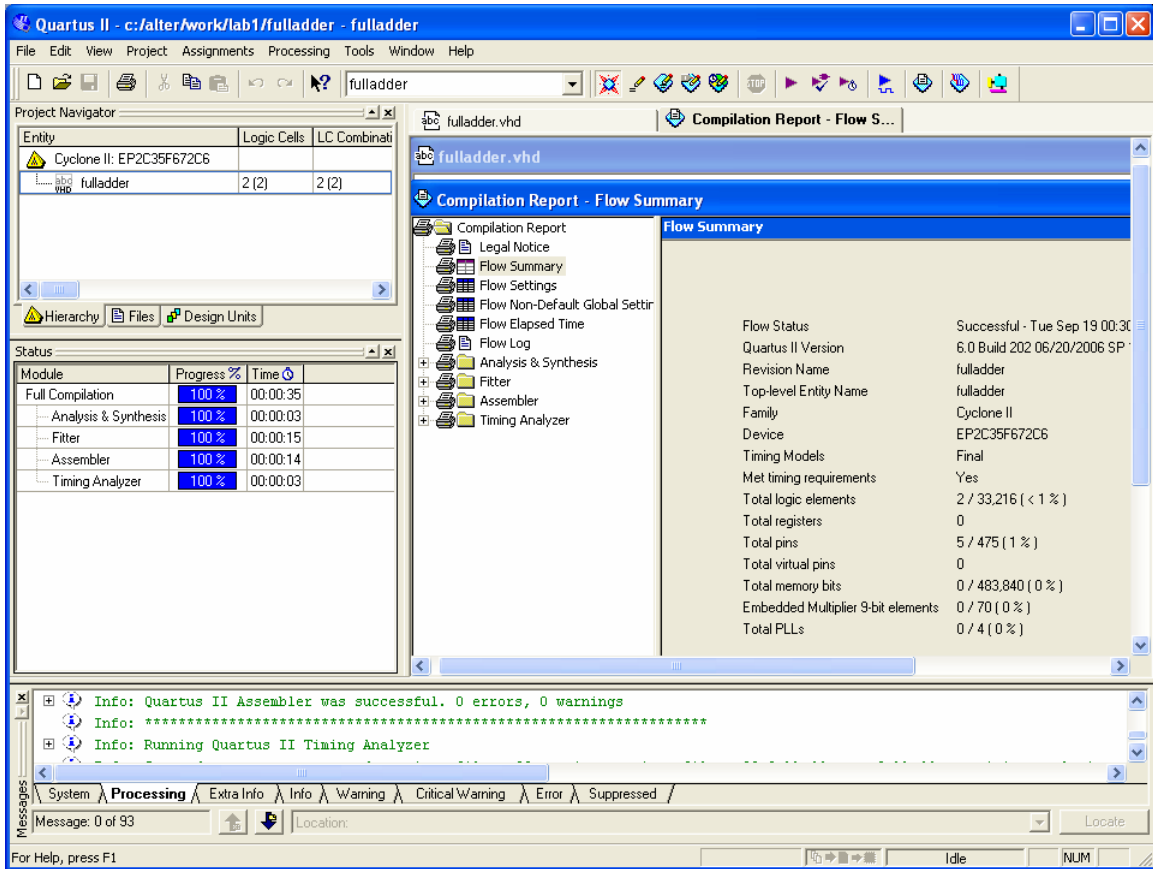


Figure 12. Display after a successful compilation.

4. Pin Assignment

The DE2 board has hardwired connections between the FPGA pins and the other components on the board. We will use two toggle switches, labeled *SW0*, *SW1* and *SW2*, to provide the external inputs, *a*, *b* and *cin*, to our example circuit. These switches are connected to the FPGA pins N25, N26 and P25, respectively. We will connect the output *sum* and *cout* to the green light-emitting diodes labeled LEDG0 and LEDG1, which is hardwired to the FPGA pin AE22 and AF22.

Pin assignments are made by using the Assignment Editor. Select **Assignments > Assignment Editor** to reach the window in Figure 13. Enter the pin assignment as shown in Figure 13. Recompile the circuit, so that it will be compiled with the correct pin assignments.

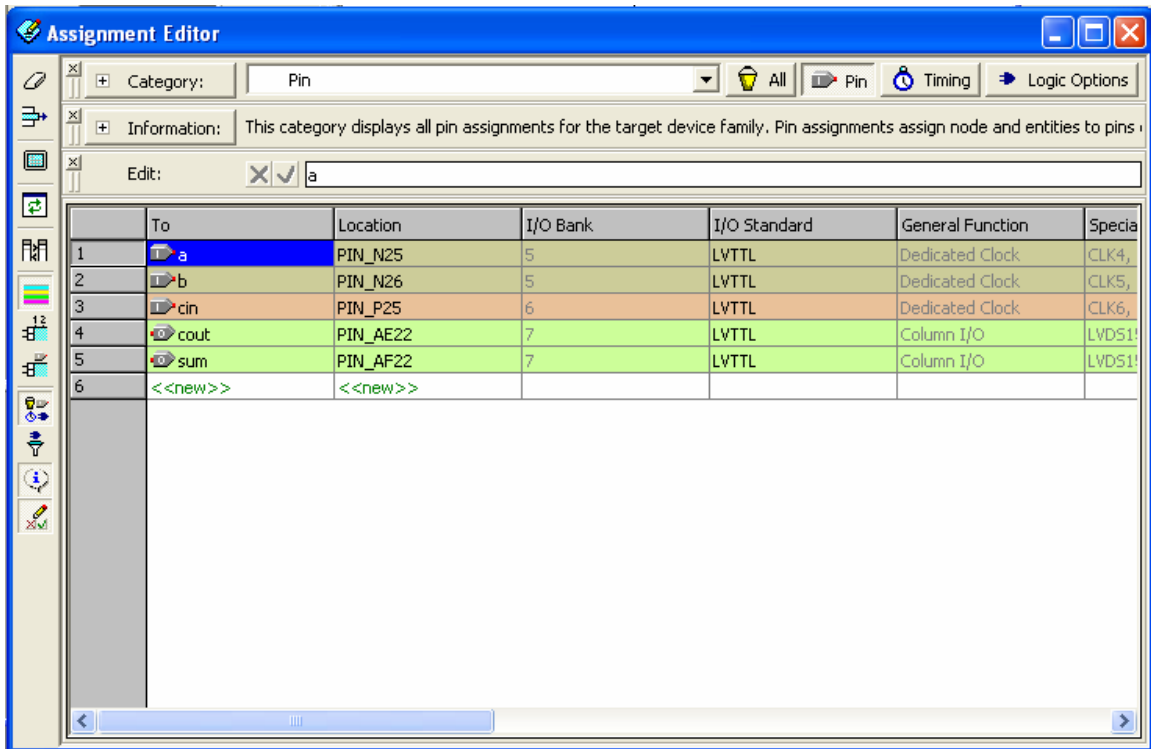


Figure 13 The Assignment Editor window.

You can import a pin assignment by choosing **Assignments > Import Assignments**. This opens the dialogue in Figure 14 to select the file to import. Type the name of the file, including the *csv* extension and the full path to the directory that holds the file, in the **File Name** box and press **OK**. Of course, you can also browse to find the desired file.

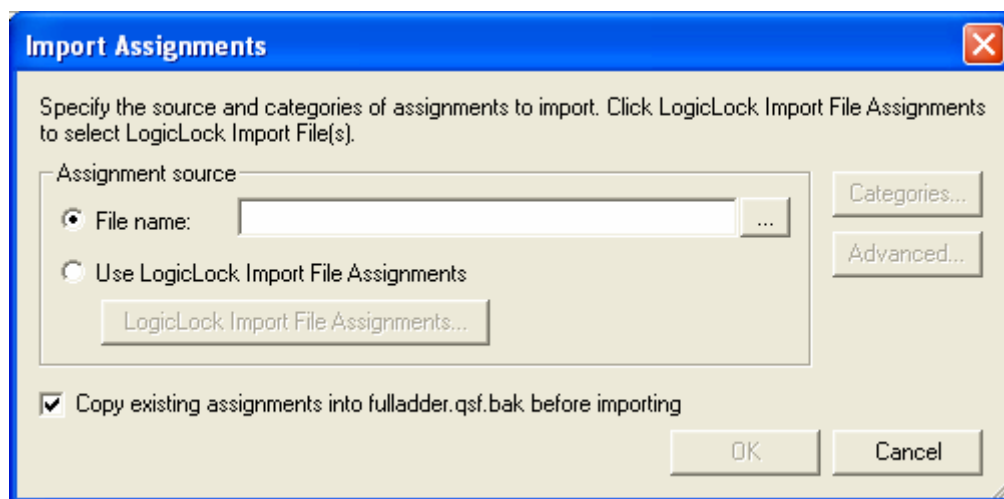


Figure 14. Importing the pin assignment.

For convenience when using large designs, all relevant pin assignments for the DE2 board are given in the file called *DE2_pin_assignments.csv*. If we wanted to make the pin assignments for our example circuit by importing this file, then we would have to use the same names in our VHDL design file; namely, SW(0), SW(1), SW(3) and LEDG(0), LEDG(1) for *a*, *b*, *cin*, *sum* and *cout*, respectively. Since these signals are specified in the

DE2_pin_assignments.csv file as elements of arrays SW and LEDG, we must refer to them in the same way in the design file. For example, in the *DE2_pin_assignments.csv* file the 18 toggle switches are called SW[17] to SW[0]; since VHDL uses parentheses rather than square brackets, these switches are referred to as SW(17) to SW(0). They can also be referred to as an array SW(17 downto 0).

5. Simulating the Designed Circuit

Before implementing the designed circuit in the FPGA chip on the DE2 board, it is prudent to simulate it to ascertain its correctness. Quartus II software includes a simulation tool that can be used to simulate the behavior of a designed circuit. Before the circuit can be simulated, it is necessary to create the desired waveforms, called *test vectors*, to represent the input signals. It is also necessary to specify which outputs, as well as possible internal points in the circuit, the designer wishes to observe. The simulator applies the test vectors to a model of the implemented circuit and determines the expected response. We will use the Quartus II Waveform Editor to draw the test vectors, as follows:

Open the Waveform Editor window by selecting **File > New**. Click on the **Other Files** tab to reach the window displayed in Figure 15. Choose **Vector Waveform File** and click **OK**.

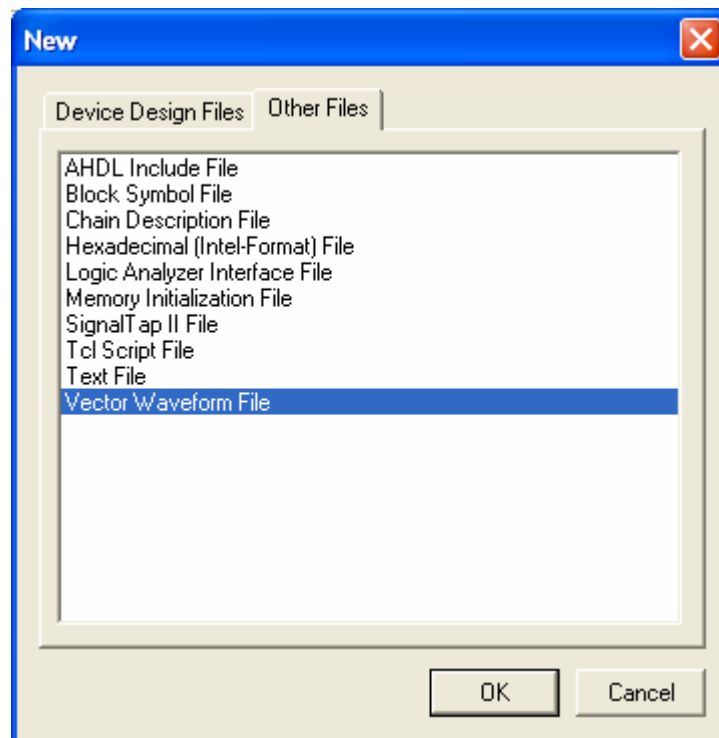


Figure 15. Choose to prepare a test-vector file.

The Waveform Editor window is depicted in Figure 16. Save the file under the name *fulladder.vwf*. Set the desired simulation to run from 0 to 20 ns by selecting **Edit > End Time** and entering 20 ns in the dialog box that pops up. Selecting **View > Fit in Window** displays the entire simulation range of 0 to 20 ns in the window.

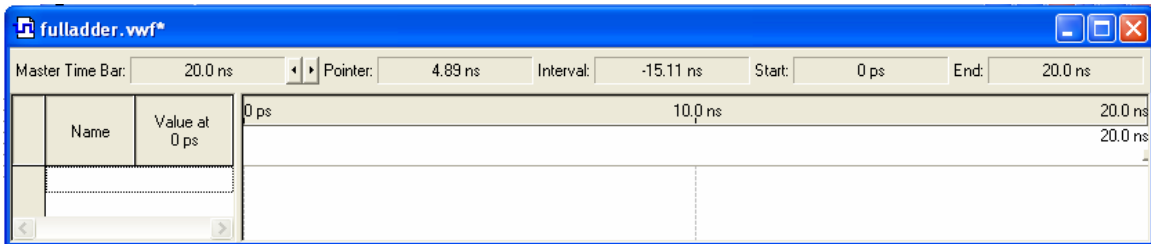


Figure 16. The Waveform Editor window.

Next, we want to include the input and output nodes of the circuit to be simulated. Click **Edit > Insert Node or Bus** to open the window in Figure 17. It is possible to type the name of a signal (pin) into the **Name** box, but it is easier to click on the button labeled **Node Finder** to open the window in Figure 18. The Node Finder utility has a filter used to indicate what type of nodes are to be found. Since we are interested in input and output pins, set the filter to **Pins: all**. Click the **List** button to find the input and output nodes as indicated on the left side of the figure. Select all signals and click the **>** sign to add it to the Selected Nodes box on the right side of the figure. Click **Ok** to close the Node Finder Window and then **Ok** in the window of Figure 17. This leaves a fully displayed Waveform Editor window, as shown in Figure 19.

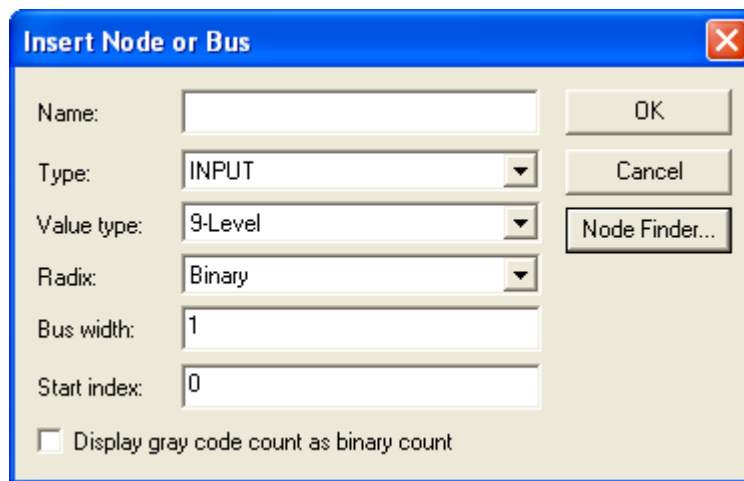


Figure 17. The Insert Node or Bus dialogue.

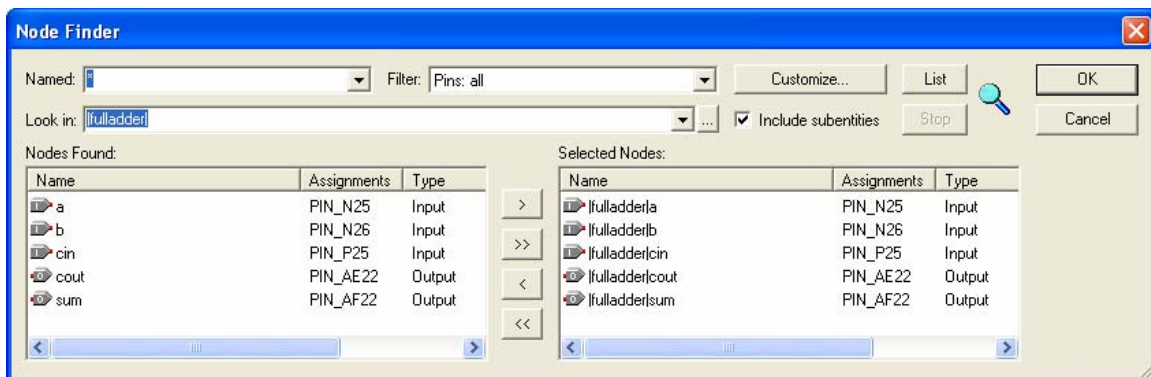
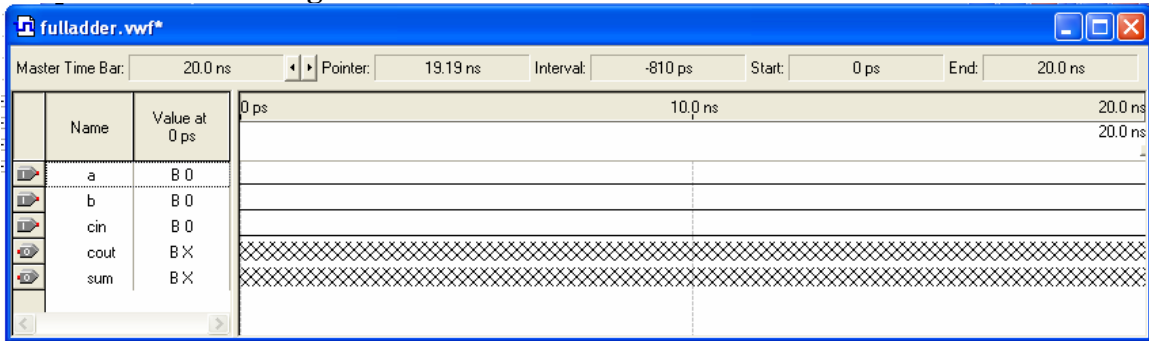




Figure 18. Selecting nodes to insert into the Waveform Editor.

Figure 19. The nodes needed for simulation.



Select signal *a* by first select the icon , then click signal *a*". Then click the icon  to bring up Figure 20 and fill in values as shown in that figure. Do the same to signal *b* and *cin* by using period of 1000ps and 2000ps respectively. Then save the file.

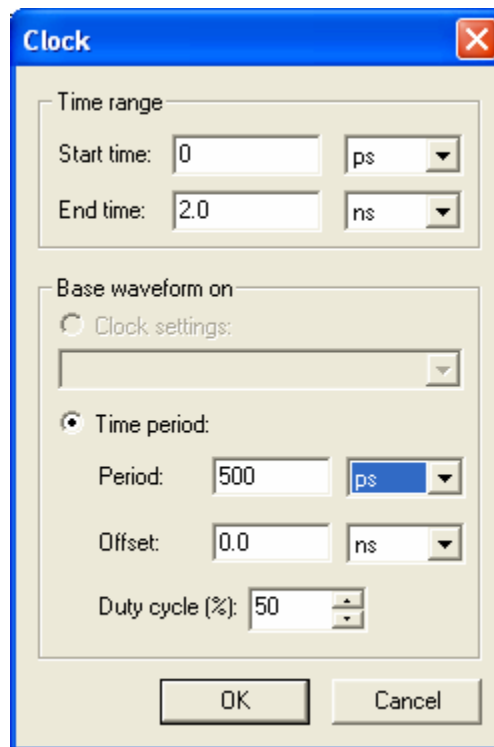



Figure 20. Clock waveform setting for *a*

A designed circuit can be simulated in two ways. The simplest way is to assume that logic elements and interconnection wires in the FPGA are perfect, thus causing no delay in propagation of signals through the circuit. This is called *functional simulation*. A more complex alternative is to take all propagation delays into account, which leads to *timing simulation*. Typically, functional simulation is used to verify the functional correctness of a circuit as it is being designed. This takes much less time, because the simulation can be performed simply by using the logic expressions that define the circuit.

To perform the functional simulation, select **Assignments > Settings** to open the Settings window. On the left side of this window click on **Simulator** to display the window in Figure 21, choose **Functional** as the simulation mode, and click **OK**. The Quartus II

simulator takes the inputs and generates the outputs defined in the *fulladder.vwf* file. Before running the functional simulation it is necessary to create the required netlist, which is done by selecting **Processing > Generate Functional Simulation Netlist**. A simulation run is started by **Processing > Start Simulation**, or by using the icon . At the end of the simulation, Quartus II software indicates its successful completion and displays a Simulation Report illustrated in Figure 22.

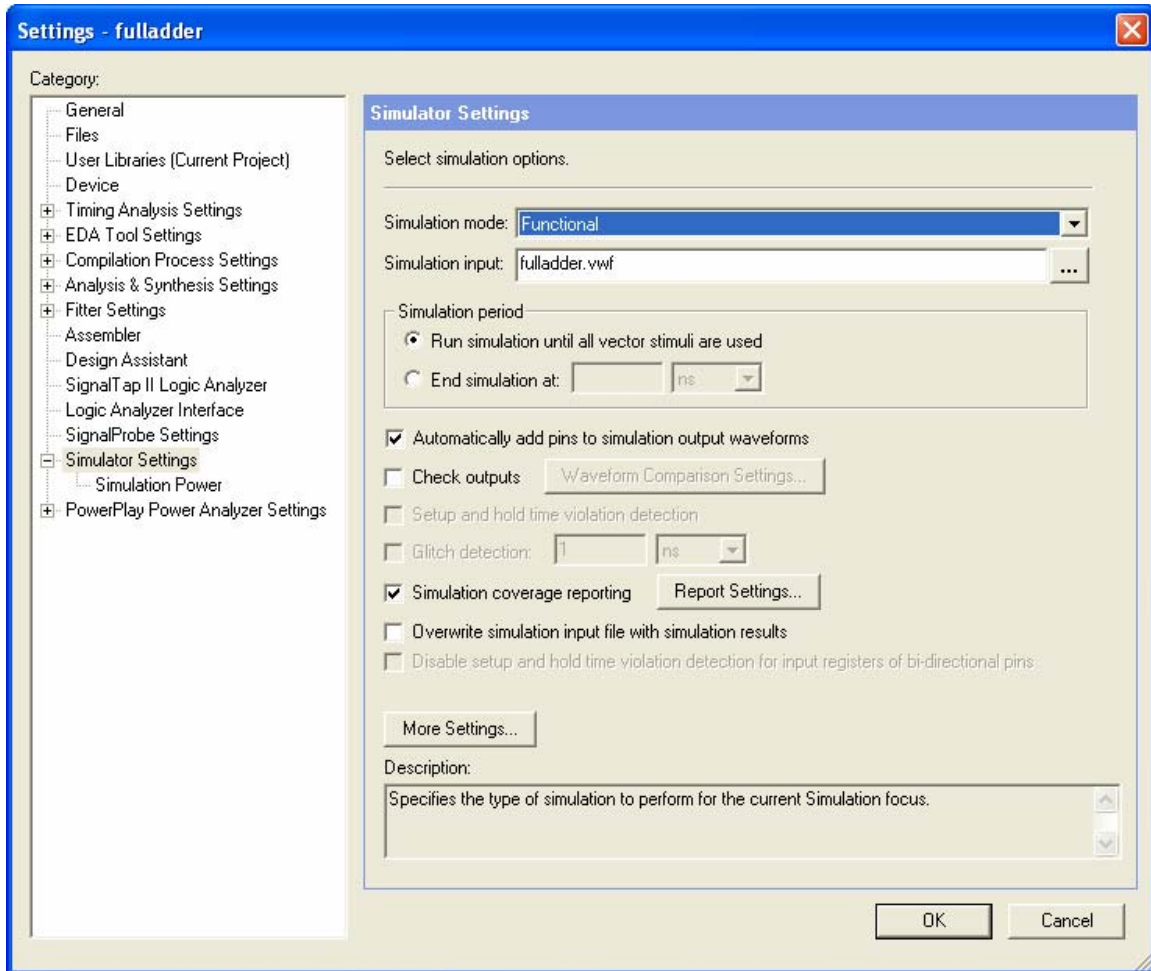


Figure 21. Specifying the simulation mode.

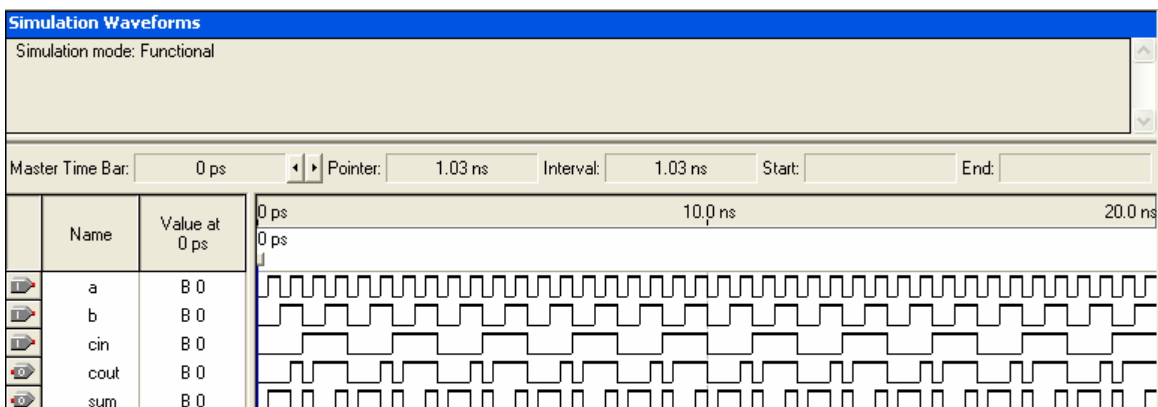


Figure 22. The result of functional simulation.

Having ascertained that the designed circuit is functionally correct, we should now perform the timing simulation to see how it will behave when it is actually implemented in the chosen FPGA device. Select **Assignments > Settings > Simulator** to get to the window in Figure 21, choose **Timing** as the simulation mode, and click **OK**. Run the simulator, which should produce the waveforms in Figure 23.

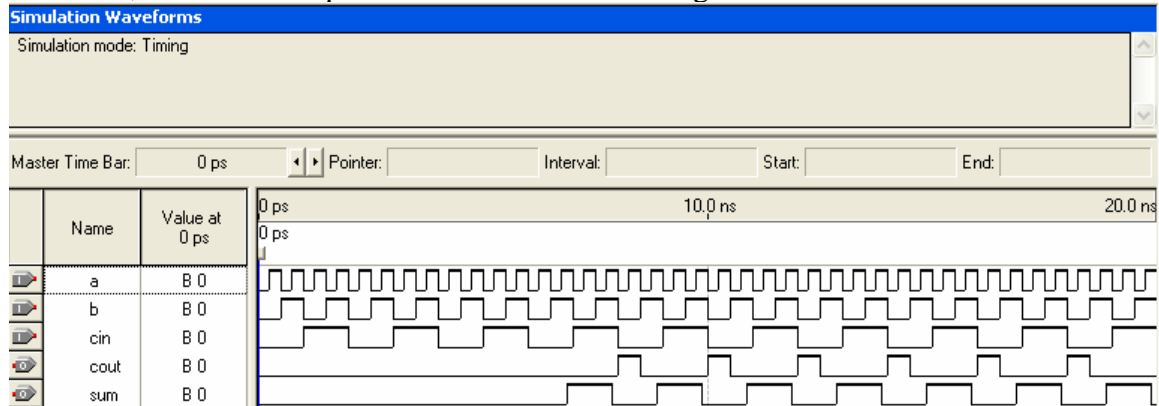


Figure 23. The result of timing simulation.

6. Programming and Configuring the FPGA Device

The programming and configuration task is performed as follows. Flip the RUN/PROG switch (on DE2 Board) into the RUN position. Select **Tools > Programmer** to reach the window in Figure 24. Here it is necessary to specify the programming hardware and the mode that should be used. If not already chosen by default, select JTAG in the Mode box. Also, if the USB-Blaster is not chosen by default, press the Hardware Setup... button and select the USB-Blaster in the window that pops up, as shown in Figure 25.

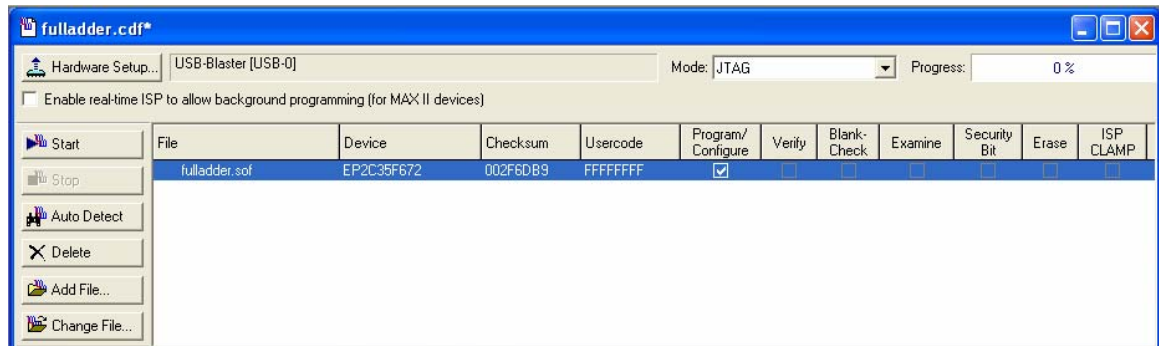


Figure 24. The Programmer window.

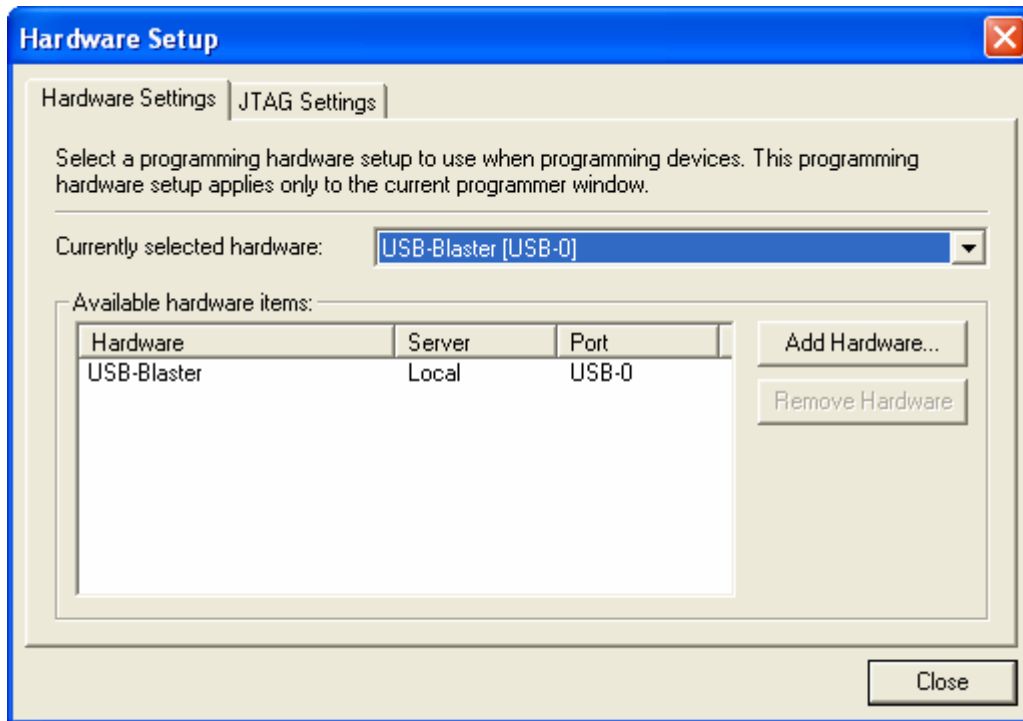


Figure 25. The updated Programmer window.

Observe that the configuration file *fulladder.sof* is listed in the window in Figure 24. If the file is not already listed, then click **Add File** and select it. This is a binary file produced by the Compiler's Assembler module, which contains the data needed to configure the FPGA device. The extension *.sof* stands for SRAM Object File. Note also that the device selected is EP2C35F672, which is the FPGA device used on the DE2 board. Click on the Program/Configure check box, as shown in Figure 24. Now, press Start in the window in Figure 24. An LED on the board will light up when the configuration data has been downloaded successfully. If you see an error reported by Quartus II software indicating that programming failed, then check to ensure that the board is properly powered on.

7. Example Project 2: Full Adder in Verilog

Follow the step 1 to create a new project but with a different name (ex: fulladder2). Click **File>New** to bring up the dialog as shown in Figure 26 and select **Verilog HDL File** and Click **OK**. Enter the code as shown in Figure 27. In Verilog, a module's inputs and outputs are listed at least twice – once in the IO list following the module name, and again inside the module where they are assigned a direction.

Verilog module outputs need to be registered. That is to say, the result of a logical expression cannot be sent directly to an output pin, but must first be buffered by a register. This is accomplished by declaring a register with the same name as the signal. Since "sum" and "cout" are output pins, add registers as shown in Figure 27. Refer to Table 1 for the Verilog syntax of common logical operators.

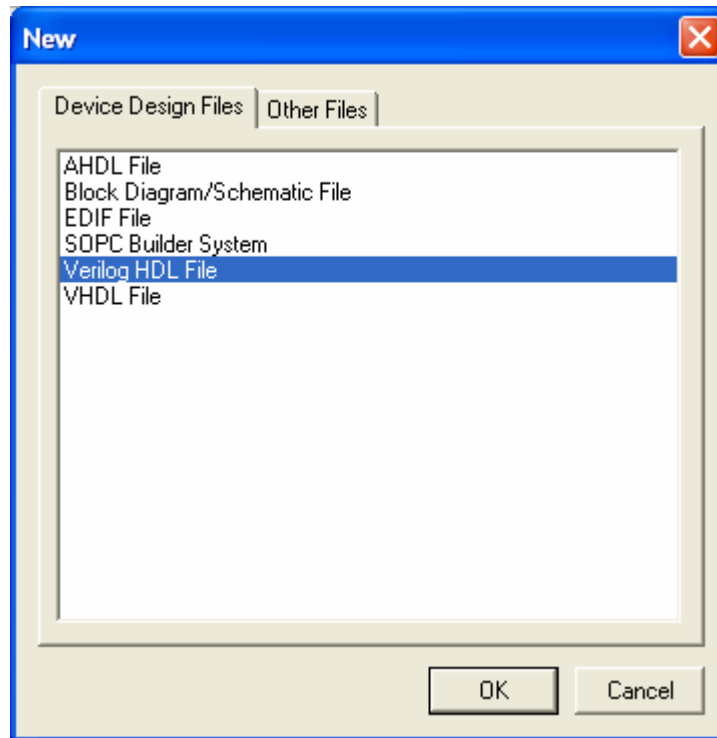


Figure 26. Create new Verilog File

```
module fulladder (a,b,cin,sum,cout) ;
    input a;
    input b;
    input cin;
    output sum;
    output cout;

    reg sum;
    reg cout;

    always @(a or b or cin)
    begin
        sum <= a ^^ b ^^ cin;
        cout <= (a && b) || (a && cin) || (b && cin);
    end
endmodule
```

Figure 27 Verilog Code

Operator	Verilog Syntax
AND	&&
OR	
XOR	^^
NOT	!

Table 1. Basic Verilog Operator

Note that the expressions for “sum” and “cout” are placed in an **always** block. An **always** block is executed any time one of the signals in the sensitivity list (“a” or b or cin” in this case) changes. This tells the synthesizer to update the “sum” and “cout” registers only when an input changes.

The procedure for synthesizing and simulating the fulladder module is the same as in the VHDL section.

8. Lab 1 Assignment

In both VHDL and Verilog, use the full-adder modules created in the above tutorials to implement four-bit adder modules with the architecture shown in figure 19. To do this, create a new source in the project where you designed the fulladder. You will have to declare multi-bit signals and instantiate the fulladder modules in this new source.

Connect the “cout” pin of each full-adder to the “cin” pin of the next.

Refer to Appendix A for module instantiation format, multi-bit signal declarations etc.

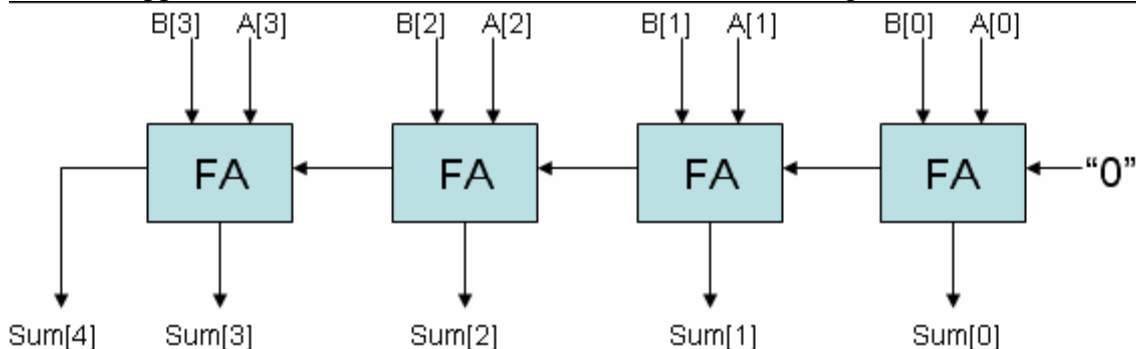


Figure 19

Once the four-bit adder is able to synthesize, run Simulator to test your design. Step the simulation with several different input combinations and verify the adder’s functionality. Record results and/or take some screenshots.

9. Lab Report Guidelines

Please write up a report on the HDL implementation and simulation of the four-bit adders created in this lab. The lab report should at least include a purpose, procedure, results, and conclusion. Please include all HDL in an appendix.

Appendix A: VHDL and Verilog Standard Formats

Standard Structure of a VHDL Design

```
entity entity_name is
    Port(signal0 : in std_logic;
          signal1 : out std_logic;
          ...
          signaln : out std_logic_vector (3 downto 0));
end entity_name;
architecture Behavioral of entity_name is
-- component declarations
component comp_name is
    Port(a : in std_logic;
          ...);
end component;
-- signal declarations
signal wire0, wire1 : std_logic;
-- main block
begin
-- behavioral and/or structural code here.
-- module instantiation
instance_name: comp_name
    port map(signal0, signal1, ...);
-- logical operations
signal3 <= (signal4 and signal5) xor signal8;
end
```

Standard Structure of a Verilog Design

```
module module_name(signal0,  
                    signal1,  
                    ... ,  
                    signaln);  
    // module signals  
    input signal0;  
    output [15:0] signal1;  
    ...  
    output signaln;  
    // internal registers  
    reg register0;  
    reg signal1;  
    // internal signals  
    wire wire0;  
    wire wire1;  
    // behavioral and/or structural code here.  
    // module instantiation  
    module_name1 instance_name1 (signal0, signal1);  
    // logical operations  
    always @ (signal4 or signal5 or signal8)  
    begin  
        signal3 <= (signal4 && signal5) ^^ signal8;  
    end  
endmodule
```